VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

LOKALIZÁCIA A PRENASLEDOVANIE NARUŠITEĽA V ZNÁMOM PROSTREDÍ

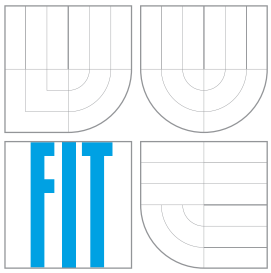
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ TOMEK

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

LOKALIZACE A PRONÁSLEDOVÁNÍ NARUŠITELE VE ZNÁMÉM PROSTŘEDÍ

LOCALIZATION AND CHASING OF AN INTRUDER IN KNOWN ENVRITONMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ TOMEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAROSLAV ROZMAN, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Tato práce koncentruje informace z oblasti algoritmů na řešení problému lokalizace a pronásledování narušitele (pursuit-evasion) ve známém prostředí. Přináší rozdělení algoritmů na základě typu a jejich zjednodušený popis. Součástí práce je také framework pro tvorbu java appletů implementujících p-e algoritmy a jejich grafickou prezentaci. V tomhle frameworku je implementován java applet pro jeden z prezentovaných algoritmů a jedna dekompozice prostředí. Vytvorený applet a informace sesbírané při tvorbě této práce jsou prezentované na vytvořených výukových stránkách.

Abstract

This thesis concentrates information from the field of algorithms for solving the problem of localization and pursuit in known environment. Division of algorithms according to their types and characteristics is given. The algorithms are described in a simplified, easier-to-understand form in comparison to detailed descriptions found in the original papers. Another part of this work is a framework for creation of java applets graphically presenting p-e algorithms. One such applet is implemented with the use of this framework. Educational web pages presenting gathered information and implemented applet are also part of the thesis.

Klíčová slova

robotika, pronásledování a lokalizace ve známém prostředí, prohledávání grafu, dekompozice prostředí

Keywords

robotics, pursuit evasion in known environment, graph search, environment decomposition

Citace

Tomáš Tomek: Lokalizácia a prenasledovanie narušiteľa v známom prostredí, bakalářská práce, Brno, FIT VUT v Brně, 2012

Lokalizácia a prenasledovanie narušiteľa v známom prostredí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Rozmana, Ph.D.

.....
Tomáš Tomek
16. května 2012

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Jaroslavovi Rozmanovi, PhD. za jeho rady při vypracovávání práce.

Obsah

Kapitola 1

Úvod

Cieľom tejto práce je zhrnutie informácií z oblasti algoritmov dostupných pre riešenie problému pursuit-evasion (p-e) v známom prostredí a ich prezentácia v ľahko zrozumiteľnej podobe. To zahŕňa:

- vytvorenie tejto textovej práce,
- implementovanie java appletu prezentujúceho jeden z algoritmov,
- pripravenie vzorových príkladov prezentujúcich funkciu algoritmu v tomto applete a
- vytvorenie výukových stránok, na ktorých budú umiestnené informácie zozbierané pri tvorbe tejto práce spolu s vytvoreným appletom.

Ďalším cieľom, ktorý je nad rámec zadania, je vytvorenie frameworku pre uľahčenie tvorby appletov prezentujúcich ďalšie p-e algoritmy.

Podstatou najčastejšie sa vyskytujúcej verzie p-e problému v známom prostredí je agent či skupina agentov - pátračov prehľadávajúcich prostredie za účelom dolapenia unikajúceho agenta či agentov - narušiteľov. Úlohou algoritmov na riešenie p-e problémov je zostavenie stratégie pre pátračov tak, aby po jej exekúcii boli chytení všetci narušitelia a prostredie bolo prehlásené za čisté.

P-e problém sa v literatúre tiež objavuje pod názvom p-e hry. Rigoróznejšia definícia problému je uvedená v časti 2.1.

V praxi sú riešenia pre tento problém aplikované napríklad na nasledovné situácie:

- zlodej unikajúci pred policajtmi,
- záchranári hľadajúci obete nešťastia,
- lokalizácia (pohybujúceho sa) tovaru v sklade,
- navádzanie rakiet,
- zachytenie rakety (protiraketová obrana),
- predchádzanie kolíziám,
- riadenie letovej prevádzky či
- odstraňovanie ropnej škvrny v mori.

V druhej kapitole sú uvedené informácie o oblasti p-e algoritmov a tabuľka sumarizujúca charakteristiky algoritmov popísaných v tejto práci. Tretia kapitola je venovaná rozličným typom dekompozícií pracovného prostredia (tá je súčasťou p-e algoritmov pracujúcich s reálnym prostredím). Štvrtá a piata kapitola popisujú jednotlivé deterministické respektíve pravdepodobnostné p-e algoritmy. V šiestej kapitole je popísaná implementačná časť práce. Výsledky práce sú zhodnotené v siedmej kapitole.

Kapitola 2

Základné informácie o problematike p-e

Táto kapitola má za úlohu uviesť čitateľa do algoritmov z oblasti prenasledovania a úniku. Prináša rigoróznejšiu definíciu p-e problému, základnú podobu p-e algoritmu, niekoľko rozdelení existujúcich algoritmov podľa vhodných charakteristík ako aj tabuľku, ktorá prehľadne sumarizuje charakteristiky algoritmov prezentovaných v tejto práci.

2.1 Definícia problému p-e

Andreas Kolling vo svojej dizertačnej práci [10] popísal p-e problém nasledovne. Pod pojmom p-e problém sa najčastejšie myslí scenár, v ktorom sa jeden či viacero pátračov snaží nájsť a chytiť jedného či viacerých narušiteľov v ohraničenom prostredí (ďalej pracovné prostredie, pp).

Keďže sa zaoberáme algoritmami pre p-e v známom prostredí, pátrači majú k dispozícii mapu pp. Nemajú ale informácie o polohe narušiteľov, ani ich stratégii. V algoritmoch pátrači teda predpokladajú takú stratégia narušiteľov, ktorá čo najdlhšie zabráni ich dolapeniu (tzv. *nepriateľskí narušitelia*). V prípade ak narušitelia takúto stratégiu nemajú, jedná sa len o zlepšenie pre pátračov. Ak by pátrači predpokladali horšiu stratégiu ako v skutočnosti narušitelia používajú, nemusela by vytvorená stratégia stačiť na dolapenie všetkých narušiteľov.

Narušitelia môžu mať ľubovoľné rozmery a rýchlosť, ale musia sa cez pp pohybovať po spojitej trase. Často majú k dispozícii mapu pracovného prostredia, stratégiu pátračov a tiež ich polohu.

Počiatočný stav p-e problému je plne kontaminované pp (teda v ktorejkoľvek jeho časti môžu byť narušitelia), v ktorom sú rozmiestnení pátrači a narušitelia. Cieľom algoritmu pre riešenie p-e problému je zostavenie takej stratégie pre pátračov, aby po jej exekúcii boli chytení všetci narušitelia a pp bolo prehlásené za čisté. Často bývajú stanovené dodatočné podmienky pre úspešnú stratégiu ako napríklad zákaz rekontaminácie.

Rekontaminácia nastáva, keď je k vyčistenej časti pp voľná cesta (t.j. nechránená pátračmi) z kontaminovanej časti pp. Keďže narušitelia môžu mať ľubovoľnú rýchlosť, nezáleží na tom, ako sú tieto časti od seba vzdialené.

Niektoré z popisovaných algoritmov môžu pracovať s inou verziou problému p-e, v takom prípade budú odlišnosti explicitne uvedené. Bežné sú odchýlky v schopnostiach pátračov

alebo narušiteľov (napríklad možnosť "teleportácie"), už spomenutý zákaz/povolenie rekontaminácie, alebo obmedzenie funkčnosti iba pre jednoducho-spojité pp.

2.2 Základná podoba p-e algoritmu

Na začiatku p-e algoritmu je pp. To môže byť reprezentované nejakou diskrétnou formou (napr. graf alebo strom), alebo mapou prostredia. V prípade mapy prostredia spravidla tvorbe prehľadávacej stratégie predchádza diskretizácia prostredia na graf alebo strom. Dôvodom je zníženie výpočtovej náročnosti algoritmu. Niektoré spôsoby diskretizácie prostredia sú popísané v kapitole 3.

Na takomto grafe potom prebieha tvorba prehľadávacej. To môže znamenať napríklad generovanie orientovaného informačného grafu, či tvorbu prehľadávacieho stromu obsahujúceho každý možný stav prehľadávania. Stratégia je potom vyjadrená ako množina prechodov medzi uzlami vytvoreného grafu či stromu. Z týchto prechodov sa následne dajú odvodiť konkrétne akcie fyzicky vykonávané robotmi reprezentujúcimi pátračov v reálnom prostredí.

2.3 Rozdelenie p-e algoritmov

Metódy riešenia p-e problémov je možné rozdeliť podľa niekoľkých kritérií. Z hľadiska reprezentácie pracovného prostredia poznáme:

1. *Diferenciálne* - spojité pp, stratégie pre pátračov a narušiteľov sú vyjadrené diferenciálnymi rovnicami a úlohou je nájsť bod rovnováhy (sedlo). V tejto rovnováhe nemôže pátrač ani narušiteľ úpravou svojej stratégie dosiahnuť lepší výsledok.

2. *Diskrétne* - pp je reprezentované grafom (prípadne stromom). Ten sa získava pomocou dekompozície. Jednou z nich je napríklad dekompozícia na základe viditeľnosti. V jej najjednoduchšej forme napríklad reprezentujú miestnosti v budove uzly grafu a hrany sú dvere medzi nimi. Viac informácií o dekompozíciách nájdete v kapitole 3.

Ďalším vhodným rozdelením p-e algoritmov je podľa princípu určovania ďalších krokov stratégie prehľadávania:

1. *Deterministické* - pátrač zvolí nasledujúcu akciu na základe exaktného výpočtu, bez použitia náhodnosti. Takéto algoritmy nezohľadňujú chybovosť senzorov (napríklad možnosť tzv. *false-negative*, t.j. neodhalenie narušiteľa aj keď sa vyskytne v zornom poli). Tiež sú, narozdiel od nasledujúcich typov algoritmov, nepoužiteľné v prípade, keď nie je dostupný dostatočný počet pátračov na vykonanie stratégie.

2. *Pravdepodobnostné* - pátrač zvolí nasledujúcu akciu na základe pravdepodobnosti výskytu narušiteľov vypočítanej podľa dostupných informácií. Takéto algoritmy niekedy zohľadňujú aj chybovosť senzorov. Vo všeobecnosti sú výpočtovo náročné a preto často nie sú použiteľné pre väčšie či komplexnejšie prostredia, ktoré vyžadujú väčší počet pátračov. Sú však použiteľné aj v prípade, keď nie je dostupný dostatočný počet pátračov na vykonanie stratégie, ktorá s určitosťou zabezpečí vyčistenie prostredia. Môže byť vytvorená stratégia, ktorá zaručí, že prostredie je čisté s nejakou pravdepodobnosťou menšou ako 1.

3. *Hybridné* - použitie pravdepodobnostného prístupu v lokálnom merítku (okolie pátrača) a deterministického v globálnom. Takéto algoritmy zabezpečujú priamu aplikovateľnosť vypočítaných stratégií do reálneho prostredia (ostatné algoritmy môžu vytvoriť stratégie garantujúce úspech v diskrétnom prostredí, no po ich pretvorení do reálneho prostredia tomu tak byť nemusí kvôli fyzickým obmedzeniam robotov).

Algoritmy pracujú s rôznou reprezentáciou prostredia:

1. *Graf* - množstvo algoritmov sa nesnaží o aplikáciu do reálneho prostredia, ale pracujú iba na grafoch. Často sa z grafov odstránením slučiek vytvárajú stromy. Dôvodom býva zjednodušenie výpočtovej náročnosti problému či zníženie počtu potrebných pátračov.

2. *2D* - takéto algoritmy sú jednoduchšie aplikovateľné do praxe. Spravidla využívajú nejakú dekompozíciu pp, podľa ktorej sa následne pp diskretizuje na graf či strom. Algoritmy často definujú obmedzenia na prostredia, na ktoré sú aplikovateľné. Napríklad nemusia zvládnuť prostredia obsahujúce "diery".

3. *2.5D a 3D* - takýchto algoritmov je z dôvodu veľkej výpočtovej náročnosti a všeobecne vyššej komplexnosti v porovnaní s vyššie spomenutými typmi veľmi málo. 2.5D predstavuje akýsi medzikrok, pracuje s tzv. výškovou mapou.

Podľa času, kedy je stratégia počítaná delíme algoritmy na:

1. *Online* - stratégia je počítaná priebežne počas prehľadávania.

2. *Offline* - stratégia je predpočítaná pred začatím prehľadávania.

Podľa typu spolupráce pátračov pri prehľadávaní:

1. *Explicitná koordinácia* - jeden pátrač môže určiť nasledujúce akcie iného. Táto koordinácia exponenciálne zvyšuje výpočtovú náročnosť stratégie.

2. *Implicitná koordinácia* - pred plánovaním nasledujúceho kroku obdrží každý pátrač informácie o predchádzajúcich akciách všetkých ostatných pátračov a zahrnie ich do výpočtu svojej nasledujúcej akcie. Tento typ koordinácie umožňuje zahrnutie spolupráce do stratégií pátračov s iba lineárnym zvýšením výpočtovej náročnosti.

3. Žiadna koordináciapátrači "vedome" nespolupracujú. To v niektorých prípadoch môže znamenať neriešiteľnosť inak s daným počtom pátračov riešiteľnej situácie

2.4 Charakteristiky popisovaných algoritmov

Podľa týchto základných charakteristík je možné zistiť, ktorý algoritmus sa hodí na ktorú konkrétnu úlohu. Napríklad nie je vhodné použiť algoritmus s rekontamináciou pri odstraňovaní ropnej škvrny v mori, kde by rekontaminácia znamenala vpustenie ropy do už vyčistenej časti mora. Rovnako nevhodné je použiť algoritmus predpokladajúci aktívne unikajúcich narušiteľov na hľadanie obetí nešťastia, ktoré sa samozrejme nesnažia aktívne unikať záchranárom.

Algoritmus	**Typ**	**Počet pátračov**	**Narušitelia**	**Optimálnosť**	**Rekontaminácia**	**Spojitosť**	**Prostredie**
Guibas	D	1	Nepriateľský	Počet pátračov	Áno	Áno	2D
Volkov	D	≥ 1	Nepriateľský	Počet pátračov	Nie	Áno	2D
Vieira	D	≥ 1	-	-	Áno	Nie	2D
Katsilieris	D	≥ 1	-	-	Nie	Áno	2D
Kehagias	D	≥ 1	Nepriateľskí	-	Áno	Nie	2D
Hollinger	D	≥ 1	Nepriateľskí	-	Áno	Nie	Graf
Moors	P	≥ 1	Nepriateľskí	-	Nie	-	2D
Strom	P	≥ 1	Nastaviteľní	-	Áno	-	2D

Tabulka 2.1: Charakteristiky popisovaných algoritmov.

Vysvetlenie pojmov použitých v tabuľke:
Počet pátračov Niektoré algoritmy sú navrhnuté iba pre jedného či dvoch pátračov a ich rozšírenie pre vyšší počet je uvedené ako tip na budúcu prácu, alebo nie je diskutované vôbec. Táto vlastnosť je dôležitá kvôli faktu, že niektoré prostredia vyžadujú na úspešné prehľadanie vyšší počet pátračov.

Optimálnosť Algoritmus môže byť optimálny či neoptimálny z viacerých hľadísk. Sú to napríklad:

- čas potrebný na prehľadanie prostredia - optimálny algoritmus nájde stratégiu prehľadávania, ktorá vyčistí prostredie za najkratší možný čas, alebo
- počet pátračov - optimálny algoritmus nájde stratégiu prehľadávania, ktorá potrebuje najmenší počet pátračov ako je s daným modelom možné.

Spojitosť Jedná sa o spojitosť dekontaminovanej časti pracovného prostredia. Niektoré algoritmy dopravia všetkých pátračov do jedného bodu pracovného prostredia a odtiaľ postupne rozširujú jeho dekontaminovanú časť. Iné svojou činnosťou vytvoria niekoľko samostatných dekontaminovaných častí, ktoré sa postupne spoja do jednej. Vo všeobecnosti sú nespojité stratégie lepšie z hľadiska počtu potrebných pátračov, keďže spojitosť je požiadavka navyše, no spojité sú jednoduchšie na výpočet.

Progresívnosť Niektoré aplikácie p-e algoritmov v praxi si vyžadujú stratégie so zakázanou rekontamináciou (napríklad spomenutý prípad s ropnou škvrnou). Progresívne algoritmy produkujú iba takéto stratégie.

Prostredie Väčšina dostupných algoritmov sa zaoberá iba reprezentáciou 2D prostredia, keďže rozšírenie do 3D značne pridáva na komplexnosti problému (možnosť schovania narušiteľa za nábytok, čiastočná viditeľnosť do vedľajších miestností cez otvorené dvere a podobne). No je dostupných aj niekoľko prác zameraných na p-e v 3D respektíve 2.5D (tzv. výšková mapa prostredia).

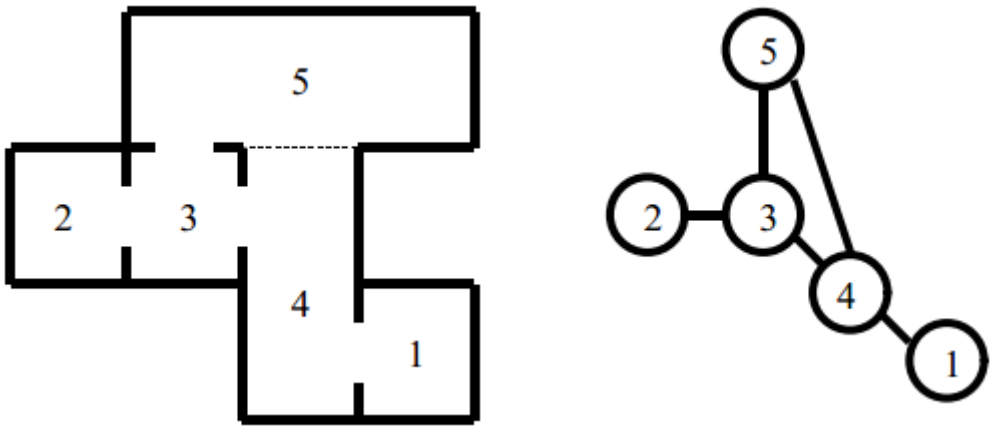
Kapitola 3

Diskretizácia pracovného prostredia

Diskretizácia pp (tiež nazvaná *dekompozícia*) je, laicky povedané, získanie grafu reprezentujúceho pp z jeho spojitej formy. Tento krok musí byť vykonaný pred použitím akéhokoľvek diskrétneho p-e algoritmu. V tejto kapitole sú stručne popísané niektoré zo spôsobov diskretizácie na graf (prípadne strom). Dodajme ešte, že vzťah medzi použitou diskretizáciou a počtom potrebných pátračov nie je ešte dostatočne dobre pochopený. Pri vyberaní vhodnej diskretizácie pre použitý model pátračov a dané prostredie je teda namieste vyskúšať ich viacero.

3.1 Miestnosti a chodby

Najjednoduchšou a najintuitívnejšou diskretizáciou je dekompozícia prostredia na základe miestností, chodieb a dverí v ňom. Miestnosti predstavujú uzly grafu a chodby prípadne dvere spájajúce miestnosti predstavujú hrany. Je zrejmé, že táto dekompozícia je použiteľná iba na vnútorné priestory budov, alebo im topologicky podobné priestory.



Obrázek 3.1: Naľavo je spojitá reprezentácia pp a napravo graf získaný použitím dekompozície miestnosti a chodby. Obrázok je prevzatý z [9].

3.2 Dekompozícia založená na kritických zmenách vo viditeľnosti

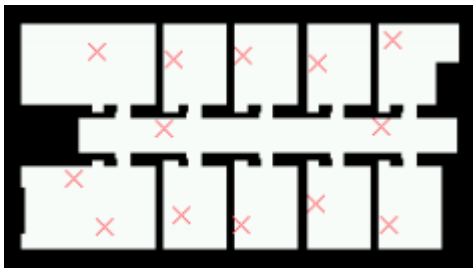
Táto dekompozícia je momentálne zrejme najpoužívanejšou (s prípadnými modifikáciami). Pracovné prostredie je pri nej rozdelené do buniek na základe kritických zmien vo viditeľnosti. Takáto zmena nastane keď do poľa viditeľnosti pátrača pribudne prekážka, alebo naopak z neho zmizne. Pri pohybe v každej takto vzniknutej bunke nenastáva z pohľadu viditeľnosti žiadna kritická zmena a je teda možné ju reprezentovať jedným uzlom grafu. Medzi dvomi uzlami je hrana, ak je možný prechod medzi nimi reprezentovanými bunkami bez nutnosti prechodu cez nejakú ďalšiu bunku.

Táto dekompozícia je použitá a podrobne popísaná pre polygonálne prostredia v práci [4]. V práci [14] ju LaValle a spol. upravili pre použitie v nepolygonálnych (zaoblených) prostrediach. Gerkey a spol. zasa prezentujú v [3] jej modifikáciu pre robotov s obmedzeným poľom viditeľnosti (konkrétne 180°).

3.3 Mrežová dekompozícia

Táto metóda začne s prázdnou množinou bodov a pridáva do nej body z pp až kým nie je každý bod pp viditeľný z aspoň jedného bodu v tejto množine. Každý bod pp je potom priradený k najbližšiemu bodu z tejto množiny. Oblasť takto pridelená každému bodu z vytvorenej množiny sa nazýva región. Každý región reprezentuje jeden uzol grafu. Medzi uzlami je hrana v prípade, ak sú k nim patriace regióny susediace. V prípade, že hranica medzi dvomi regiónmi nie je súvislá, je pridaná hrana pre každú súvislú časť hranice.

Vyberanie bodov vkladaných do množiny môže byť náhodné, alebo ovládané nejakým optimálnejším algoritmom, ktorý zabezpečí menší výsledný počet bodov v množine. Táto dekompozícia je použitá napríklad v práci [15].

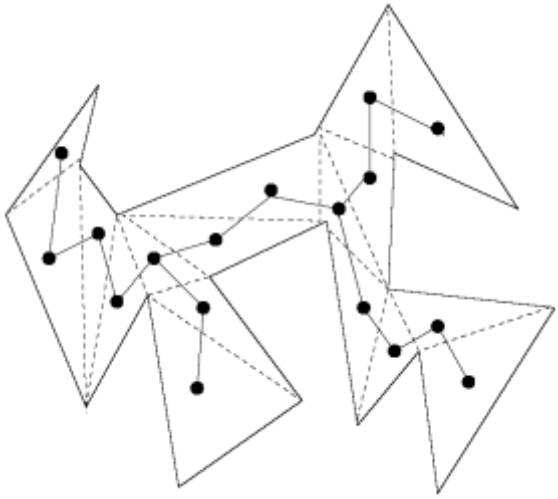


Obrázek 3.2: Množina bodov náhodne umiestnených do prostredia tak, aby bol každý jeho bod viditeľný z aspoň jedného bodu množiny. Obrázok je prevzatý z [15].

3.4 Triangulácia

Triangulácia polygónu je rozloženie polygónu maximálnym počtom nepretínajúcich sa uhlopriečok. Každý takto vzniknutý trojuholník je v grafe reprezentovaný jedným uzlom. Medzi dvomi uzlami je hrana, ak nimi reprezentované trojuholníky majú spoločnú stranu. Graf takto vzniknutý z jednoducho-spojitého prostredia je vždy strom.

Táto dekompozícia je použitá napríklad v prácach [7] a [8].



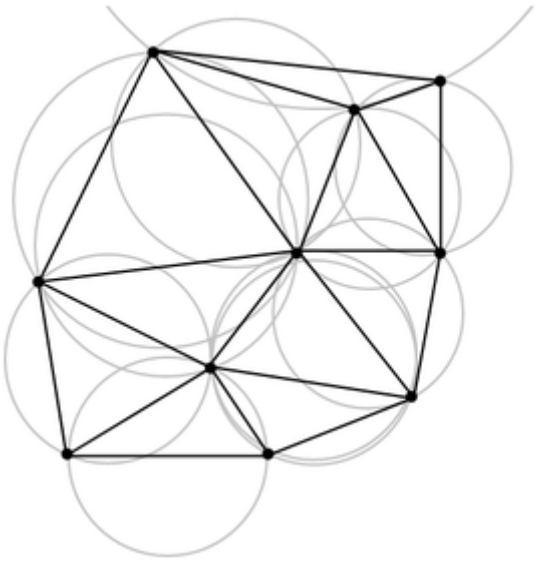
Obrázek 3.3: Triangulácia prostredia a z nej vytvorený strom. Obrázok je prevzatý z [7] .

3.5 Obmedzená Delaunayho triangulácia

Táto triangulácia dovoľuje vynútiť generovanie trojuholníkov s obmedzenou veľkosťou uhlov a strán. To umožňuje zahrnúť do diskretizácie prostredia obmedzenia na dosah senzorov pátračov.

Delaunayho triangulácia pre množinu bodov v rovine je taká triangulácia, že ani jeden z bodov v tejto množine neleží vo vnútri kružnice opísanej ktorémukoľvek z trojuholníkov.

Nejaké myšlienky ohľadom tejto triangulácie uviedli vo svojej práci Hollinger a spol. [6]. Ďalšie informácie o Delaunayho triangulácii je možné nájsť v [18].



Obrázek 3.4: Delaunayho triangulácia so zobrazenými opísanými kružnicami. Obrázok je prevzatý z wikipedie.

3.6 Dekompozícia pomocou stredovej osi

Stredová os (alebo kostra) prostredia je definovaná ako krivka vytvorená stredmi maximálnych kružníc vpísaných do pp. Z tejto kostry sa vytvorí topologický strom. Topologický strom je orientovaný strom s koreňom v *počiatočnom uzle* (uzol v ktorom začínajú všetci pátrači) Uzlami tohto stromu sú koncové body a rázcestia v kostre, hrany zase reprezentujú úseky spájajúce tieto body.

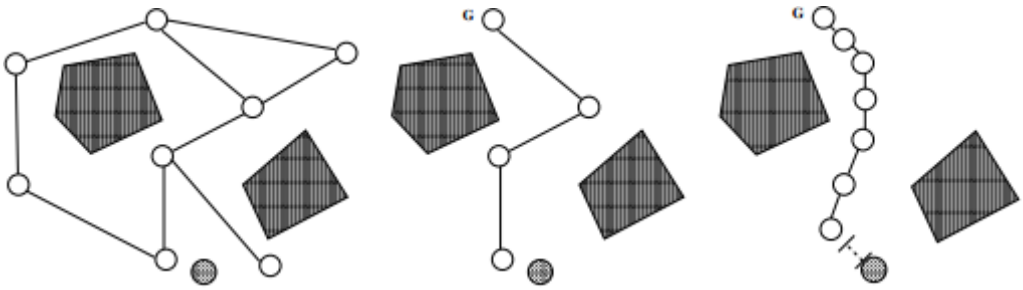
Táto dekompozícia je podrobnejšie popísaná a použitá v [21].

Obrázek 3.5: Na obrázku vľavo vidíme pracovné prostredie a na obrázku vpravo je jeho kostra. Obrázok je prevzatý z [21].

3.7 Roadmapy

Roadmapa je graf reprezentujúci možné nekolízne cesty prostredím. Môže byť generovaná rôznymi algoritmami. Najjednoduchší z nich náhodne najprv zvolí niekoľko bodov v pp a následne sa ich snaží pospájať tak, aby vyhovovali podmienkam nekolíznosti. Po zvolení nejakej konkrétnej cesty z roadmapy robotom môže byť táto ešte optimalizovaná pre urýchlenie jej prejdenia.

Táto diskretizácia je použitá napríklad v [17]. Viac informácií o konštrukcii roadmáp je možné nájsť v práci [12].



(a) Príklad roadmapy (b) Cesta vybratá robotom (c) Optimalizovaná cesta

Obrázek 3.6: Na obrázku (a) sú pospájané náhodne vybraté body v prostredí. Na obrázku (b) je jedna robotom vybratá cesta a na obrázku (c) je táto cesta optimalizovaná pre rýchlejší prechod. Obrázky sú prebraté z [17].

Kapitola 4

Deterministické p-e algoritmy

V tejto kapitole sú popísané algoritmy, v ktorých pátrač volí nasledujúcu akciu na základe exaktného výpočtu, bez použitia náhodnosti. Jednotlivé časti tejto a nasledujúcej kapitoly majú názvy zložené z mena hlavného autora práce v ktorej bol uverejnený popisovaný algoritmus a prípadných charakteristík algoritmu. Tento prístup som zvolil kvôli nemožnosti stručného a zároveň výstižného pomenovania niektorých algoritmov.

4.1 Guibas a spol.

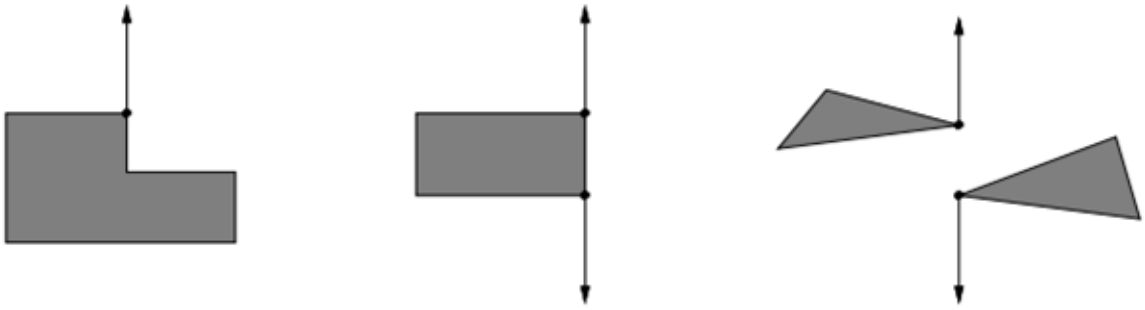
Guibas a spol. vo svojej práci prezentujú [4] algoritmus pre jedného pátrača. Funguje teda iba pre jednoducho-spojité pp, keďže v prípade prostredí "s dierami" sú nutní minimálne dvaja pátrači (ak sa narušiteľ schová za prekážku, cez ktorú pátrač nevidí, tak vždy existuje taká stratégia pre narušiteľa, aby garantovala nekonečný únik). Vypočítané stratégie v niektorých pp vyžadujú rekontamináciu. Základnou myšlienkou algoritmu je rozložiť pp do konzervatívnych častí, zostaviť orientovaný graf na základe informačných prechodov medzi týmito časťami a na ňom exekvovať prehľadávanie.

4.1.1 Stručný prehľad

1. Dekompozícia pp do konzervatívnych častí
2. Vytvorenie grafu na základe vzťahov konzervatívnych častí
3. Vytvorenie informačného grafu
4. Hľadanie cesty v informačnom grafe

4.1.2 Dekompozícia pracovného prostredia

Nech $\mathcal{D}$ reprezentuje súbor konvexných oblastí získaných rozložením pp (ďalej bude pp pre lepšiu prehľadnosť označované aj ako F). Hranice týchto oblastí získame predĺžením hrán v pp a určitých párov bodov ako na obrázku 4.1. Každá hrana je predĺžená do všetkých možných smerov. Každý pár bodov je predĺžený smerom von, a to iba v prípade, že oba smery sú voľné.

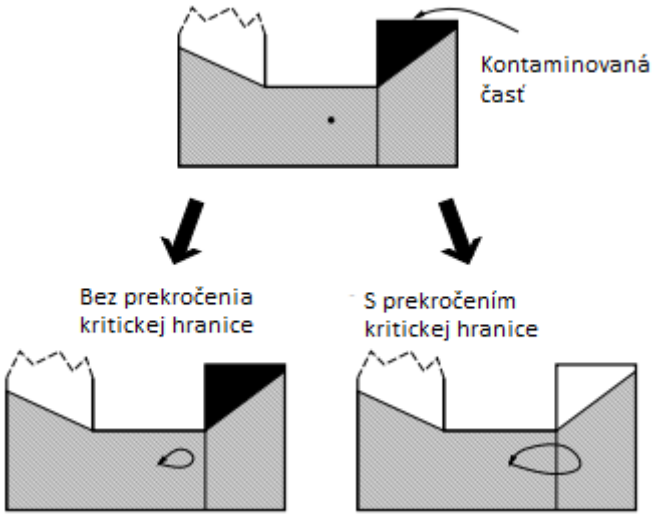


Obrázek 4.1: Tri možné prípady predlžovania hrán pri vytváraní konzervatívnych častí pp. Obrázok je prebratý z [4].

Všetky oblasti vytvorené hranami pp a týmito predĺženiami sú konzervatívne. To znamená, že pri pohybe v jednej oblasti nenastane kritická zmena vo viditeľnosti (dôkaz viď lemma 4 v [4]).

Pred pokračovaním v popise častí algoritmu si musíme definovať informačný stav.

Informačný stav Nech $V(q)$ je polygón viditeľnosti pátrača na pozícii $q \in F$, kde F je množina bodov patriaca do pp. Vo $V(q)$ môžu byť dva typy hrán, a to hrany patriace do hraníc F a hrany križujúce vnútro F. Druhé menované budeme nazývať *interiérové hrany*. Každá takáto hrana bude mať priradené označenie - buď "1" alebo "0". Hrana bude označená "1" ak oddeľuje $V(q)$ od kontaminovanej časti F a naopak "0" v prípade, že ho oddeľuje od čistej časti F. Nech $B(q)$ symbolizuje binárny vektor týchto označení hrán, v ktorom je každá interiérová hrana vo $V(q)$ reprezentovaná jednou značkou. Potom pár $(q, B(q))$ jedinečne charakterizuje jeden informačný stav.



Obrázek 4.2: Na ľavom spodnom obrázku vidíme pohyb pátrača v rámci jednej konzervatívnej časti, nenastane teda kritická zmena viditeľnosti a po návrate pátrača na pôvodnú pozíciu je ten v rovnakom informačnom stave. Na obrázku vpravo dole pátrač prejde do inej konzervatívnej časti a keď sa vráti na pôvodnú pozíciu, je v inom informačnom stave (interiérová hrana v pravej časti pp sa zmení z kontaminovanej na čistú). Obrázky sú prevzaté z [4]

4.1.3 Vytvorenie grafu reprezentujúceho pracovné prostredie

Oblasti v $\mathcal{D}$ a ich susednosť v pp definujú graf G. Každý uzol tohto grafu zodpovedá jednej oblasti $D \in \mathcal{D}$ a medzi dvomi uzlami je hrana vtedy, ak je možný prechod z oblasti reprezentovanej prvým uzlom do oblasti reprezentovanej druhým bez nutnosti prechodu cez ďalšie oblasti. Cesta v pp môže byť podľa cesty v G vytvorená spojením ťažísk dvoch susediacich oblastí v $\mathcal{D}$ lineárnym segmentom. Cieľom pátrača je nájsť takú cestu v G, aby v istom momente boli všetky interiérové hrany čisté. Je možné, že počas tejto cesty bude musieť pátrač jednotlivé uzly G navštíviť viackrát, pretože vektor označení hrán $B(q)$ sa môže vždy líšiť. Na začiatku cesty je pátrač na nejakej pozícii so všetkými interiérovými hranami označenými "1" a na jej konci je na nejakej pozícii so všetkými interiérovými hranami označenými "0".

4.1.4 Vytvorenie informačného grafu

Informačný graf je orientovaný graf, v ktorom uzly reprezentujú informačné stavy (ktoré sme si definovali vyššie) a hrany sú prechody medzi nimi.

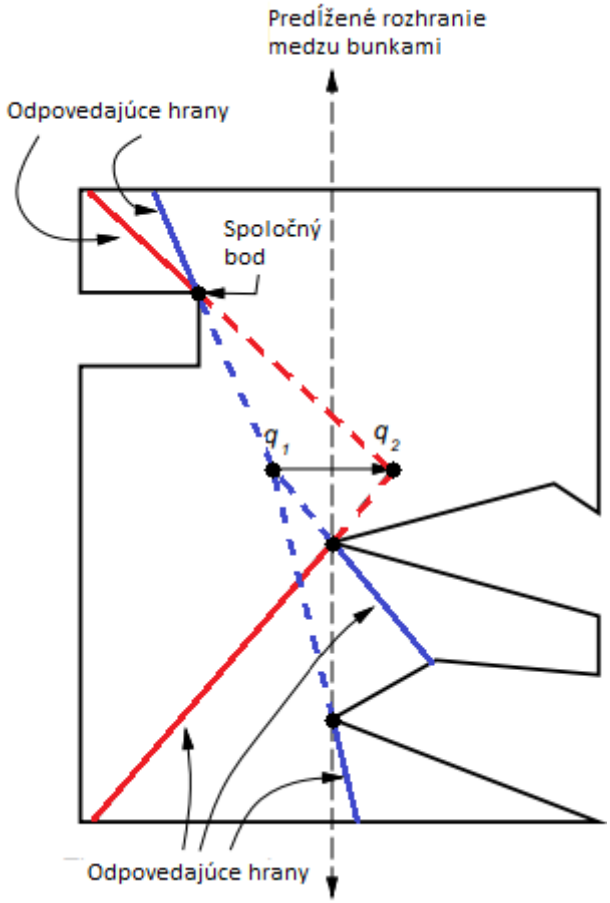
Odvodenie informačného grafu G_I z grafu G Každý uzol G odpovedá niekoľkým uzlom v G_I, jeden pre každú možnú hodnotu $B(q)$. Jeden uzol v G_I zodpovedá jednému informačnému stavu a je teda jedinečne charakterizovaný párom $(q,\ B(q))$.

Kvôli definícii hrán v G_I si musíme najprv zmapovať, čo sa môže diať s interiérovými hranami pri prechode z jednej oblasti do druhej. Rozlišujeme štyri prípady:

- *zmiznutie* interiérovej hrany,
- *objavenie* interiérovej hrany - vždy dostane označenie "0",
- *zlúčenie* jednej alebo viacerých interiérových hrán do jednej - dostane označenie "1", ak aspoň jedna zo zlučovaných hrán mala označenie "1", inak dostane označenie "0" a
- *rozdelenie* jednej interiérovej hrany na dve alebo viacero hrán - nové hrany dostanú označenie ako mala pôvodná hrana.

Na základe zmienených štyroch pravidiel a toho, ako sa mení $V(q)$ pri prechode medzi oblasťami sú vytvorené hrany medzi jednotlivými informačnými stavmi.

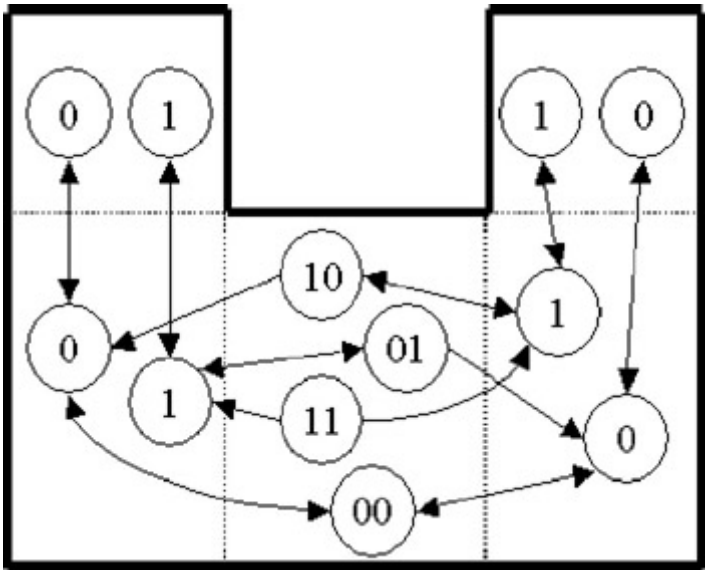
Na obrázku 4.3 vidíme presun pátrača z pozície q_1 na pozíciu q_2, pri ktorej nastáva prechod rozhraním oblastí. V spodnej časti vidíme zlúčenie dvoch modrých hrán do jednej červenej. Z pohľadu hrany v hornej časti obrázku kritická zmena nenastáva.



Obrázek 4.3: Vývin hrán pri prechode pátrača z jednej oblasti do druhej. Obrázok je prevzatý z [4].

4.1.5 Hľadanie cesty v informačnom grafe

Cestu v G_I nájdeme použitím ľubovoľného algoritmu pre hľadanie cesty v grafe. Príkladom je použitie Dijkstrovho algoritmu pre nájdenie najkratšej cesty s použitím vzdialenosti ťažísk konzervatívnych častí ako váhy hrany. Takto vypočítaná cesta v skutočnom prostredí by vo väčšine prípadov bola "hranatá", takže by bolo na mieste použiť vyhladzovací algoritmus.



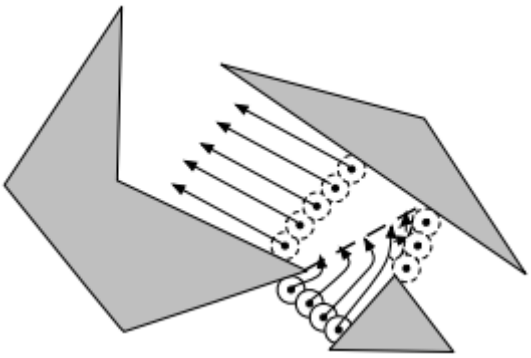
Obrázek 4.4: Grafické znázornenie informačného grafu. Obrázok je prevzatý z [2].

4.1.6 Rozšírenie pre viacero pátračov

Veľkou prekážkou pre rozšírenie prezentovaného algoritmu pre viacero pátračov je fakt, že môže nastať kritická zmena vo viditeľnosti aj keď žiadny pátrač neprekročí hranicu konzervatívnej časti (príklad takejto situácie je možné vidieť na obrázku 10 v [4]. To podstatne zvyšuje implementačnú náročnosť algoritmu a zároveň aj počet konzervatívnych častí, čo zase dramaticky zvyšuje výpočtovú náročnosť algoritmu.

4.2 Frontier-based prehľadávanie bez rekontaminácie

Volkov a spol. vo svojej práci [21] prezentujú spôsob prehľadávania, pri ktorom sa pátrači pohybujú prostredím vo formácii, ktorá oddeľuje vyčistenú časť prostredia od kontaminovanej. Pátrači teda tvoria akúsi hranicu (frontier), viď obrázok 4.5. Algoritmus pracuje iba s prostrediami "bez dier". Rozšírenie na prostredia s dierami je uvádzané ako vhodná téma na prácu do budúcnosti.



Obrázek 4.5: Formácia, v ktorej sa pohybujú pátrači. Obrázok je prevzatý z [11].

4.2.1 Stručný prehľad

1. Tvorba kostry prostredia
2. Tvorba topologického stromu z kostry
3. Hľadanie cesty v topologickom strome

4.2.2 Dekompozícia pp na topologický strom

Diskretizácia pp je vykonaná dekompozíciou pomocou stredovej osi (popísaná tu 3.6). Pripomeňme si, že topologický strom je orientovaný strom s koreňom v *počiatočnom uzle* (uzol v ktorom začínajú všetci pátrači). Tento strom sa tvorí na základe tzv. *kostry* prostredia. Pre každý uzol s je definovaný atribút $\gamma(s)$, ktorého hodnota je počet hrán vychádzajúcich z s. Každá hrana e má pridelenú váhu $\alpha(e)$. Tá určuje počet pátračov potrebných na prechod z uzlu na jednom konci hrany, do uzlu na konci druhom. Bližšie informácie o týchto atribútoch a o ich výpočte sú uvedené v [21], kapitola 4.

4.2.3 Algoritmus prechádzania topologického stromu

Prechádzanie topologického stromu berieme ako hru. V tejto hre má každý uzol stromu pridelenú značku λ, ktorá môže nadobúdať hodnoty *kontaminovaný*, *skúmaný* a *čistý*. Na jej začiatku sú všetci pátrači v počiatočnom uzle, ktorý má značku skúmaný. Počiatočný uzol je zvolený ľubovoľne z množiny všetkých koncových uzlov. Všetky uzly okrem počiatočného majú na začiatku značku kontaminovaný. Označenie všetkých uzlov spolu s počtom pátračov v nich umiestnených definuje *stav hry*. Súbor prechodov medzi takýmito stavmi nazývame *prehľadávacia stratégia*.

Každý takýto prechod je jedno kolo hry, v ktorom sa pátrači presúvajú (narušitelia sa presúvajú tiež, no ich stratégia nás nezaujíma). V [21], časť 4.1 sú uvedené pravidlá, ktoré pátrači musia dodržiavať pri presune. Tieto pravidlá mimo iné zabraňujú rekontaminácii a definujú, ako sa pátrači správajú pri rozdeľovaní na skupiny či naopak pri zlučovaní skupín.

Hru vyhrajú pátrači, ak sú v istom čase čisté všetky uzly. Narušitelia vyhrajú, ak sú všetci pátrači uviaznutí a existuje aspoň jeden uzol, ktorý nie je čistý.

Nasleduje rekurzívny algoritmus, ktorým sa riadia pátrači pri prechádzaní topologického stromu. Je optimálny z pohľadu počtu potrebných pátračov. Algoritmus je prevzatý z [21].

Algoritmus 1 $Clear(T(s), n_0)$

Majme topologický strom T s koreňom v uzle s. Hra je v počiatočnom stave.
if s je list **or** všetci potomkovia s sú označení ČISTÝ **then**
  označ $s \leftarrow$ ČISTÝ
  Vráť sa do nadradeného uzlu. Ak neexistuje, ukonči sa.
end if
if $\lambda(s) =$ KONTAMINOVANÝ **then**
  označ $s \leftarrow$ SKÚMANÝ
end if
if $\gamma(s) = 1$ **then**
  $Clear(T(p_1), n_0)$
end if
if $\gamma(s) > 1$ **and** všetci potomkovia s sú označení KONTAMINOVANÝ **then**
  for $i = 2$ **to** $i = \gamma(s)$ **do**
    $Clear(T(p_i), \alpha(e_i(s)))$
  end for
  $Clear(T(p_i), n_0 - \sum_{i=2}^{\gamma(s)} \alpha(e_i(s)))$
end if
if $\gamma(s) > 1$ **and** niektorí potomkovia s nie sú označení KONTAMINOVANÝ **then**
  nech $i = min_{2,\ldots,\gamma(s)}\{i|\lambda(p_i) \neq$ ČISTÝ$\}$
  $Clear(T(p_i), n_0)$
end if

4.3 Rozdelenie p-e problému na viacero problémov s jedným narušiteľom

Algoritmus [20] prezentovaný Vieirom a spol. pracuje s mierne odlišným p-e problémom. Prvým rozdielom je, že všetci účastníci p-e hry majú informácie o aktuálnej polohe všetkých ostatných účastníkov (ktoré ale nemusia byť vždy presné). Druhý rozdiel je v rýchlosti pohybu - je rovnaká pre pátračov a narušiteľov. Obaja sa môžu pohnúť iba o jeden uzol grafu za jeden časový krok. Posledným rozdielom je, že účastníci hry sa nehýbu naraz, ale najprv sa pohnú všetci pátrači, následne všetci narušitelia, potom opäť pátrači a tak ďalej, až kým nenastane koniec hry.

Koniec hry nastane buď chytením všetkých narušiteľov (výhra pátračov), alebo ak algoritmus zistí, že sa cyklicky opakuje sekvencia stavov (výhra narušiteľov).

4.3.1 Stručný prehľad

1. Predpočítanie diagramu prechodov medzi stavmi pre jednu ľubovoľnú c(G)-1 pod-hru
2. Rozdelenie hry na viacero c(G)-1 hier
3. Informovanie každého účastníka hry o pozícii všetkých ostatných účastníkov
4. Výpočet rozdeľovacieho algoritmu každým pátračom samostatne (čím je pridelený konkrétnemu narušiteľovi)
5. Hranie hry za použitia predpočítaného prechodového diagramu a aktualizácií ohľadom pozícií účastníkov až do jej skončenia

4.3.2 Diagram prechodov medzi stavmi

Ide o orientovaný graf, znázorňujúci priebeh hry. Počiatočný stav je definovaný pozíciami pátračov a narušiteľov na začiatku hry. Od tohto stavu sú vedené hrany k všetkým stavom, ktoré môžu vzniknúť pohybom pátračov. Z týchto stavov sú zase vedené hrany do všetkých stavov, ktoré môžu vzniknúť pohybom narušiteľov. Takto pokračujeme v generovaní grafu, až kým nenastane stav, v ktorom sú všetci narušitelia dolapení, pričom ak nejaký pohyb vedie k stavu, ktorý už v hre niekedy nastal, nie je vytvorený nový stav v grafe, ale hrana je vedená k tomuto stavu. Pre všetky stavy môžeme potom vypočítať cenu - počet kôl potrebných pre dosiahnutie stavu, v ktorom sú všetci narušitelia dolapení použitím metódy zdola-nahor.

Stratégia pre účastníkov hry je potom definovaná algoritmom minimax:

- *Pátrači* si vždy vyberú nasledujúci stav tak, aby jeho cena bola čo najmenšia.
- *Narušitelia* si vždy vyberú nasledujúci stav tak, aby jeho cena bola čo najväčšia.

4.3.3 Rozdelenie narušiteľov medzi pátračov

Z dôvodu výpočtovej náročnosti diagramu prechodov medzi stavmi pre všetkých narušiteľov a pátračov naraz, je každému narušiteľovi pridelený jeden tím pátračov a až pre tieto "pod-hry" je diagram počítaný. Algoritmus teda predpokladá, že celkový počet pátračov p je väčší alebo rovný minimálnemu počtu pátračov potrebných pre dolapenie jedného narušiteľa v danom prostredí $c(G)$, vynásobeného počtom narušiteľov r.

$$p \geq c(G) * r$$

Ak je p väčšie, nadbytoční pátrači sú náhodne pridelení k tímom, čím vzniká redundancia užitočná v prípade, že nejaký robot zlyhá.

Algoritmus generovania rozdelenia pátračov do tímov a ich pridelenie k narušiteľom funguje nasledovne. Najprv je vypočítaný diagram prechodov medzi stavmi pre jednu ľubovoľnú c(G)-1 pod-hru. Keďže ten obsahuje všetky možné ťahy, obsahuje aj všetky možné stavy dosiahnuteľné v c(G)-1 hre, a teda aj všetky počiatočné stavy hry, ktoré môžu nastať kombináciou ľubovoľného tímu pátračov a narušiteľa. Preto stačí vypočítať jediný takýto diagram a pre ostatné hry v ňom iba vyhľadať počiatočný stav.

Potom sa vygenerujú všetky možné kombinácie pátračov do tímov, pričom jeden pátrač musí patriť naraz iba do jedného tímu. Pre každé takéto rozdelenie pátračov sa vypočíta cena. Táto cena je rovná minimálnej cene spomedzi všetkých možných pridelení narušiteľov tímom pátračov. Cena jedného pridelenia narušiteľov je zase rovná maximálnej cene chytenia narušiteľa v danom pridelení narušiteľov. Ak je cena aktuálneho rozdelenia pátračov lepšia ako doteraz najlepšia, zaznamená sa aj spolu s rozdelením pátračov. Po prejdení všetkých konfigurácií tímov je vrátená najlepšia konfigurácia a podľa nej sú pátrači rozdelení do tímov a k nim pridelení narušitelia. Tohto rozdelenia sa pátrači držia po celú dobu hry.

Algoritmus 2 Zjednodušený algoritmus rozdelenia narušiteľov do tímov a ich pridelenia k narušiteľom.

1: Vypočítanie diagramu prechodov medzi stavmi pre jednu c(G)-1 pod-hru
2: Vygenerovanie všetkých možných kombinácií tímov tak, že jeden pátrač môže zároveň patriť iba do jedného tímu.
3: Pre všetky konfigurácie tímov:
4: Vypočítaj najnižšiu cenu dosiahnuteľnú pri danej konfigurácii tímov.
5: Aktualizuj konfiguráciu z najnižšou cenou.
6: Vráť konfiguráciu s najnižšou cenou.

4.4 Prehľadávanie do hĺbky a súčasné prehľadávanie

Katsilieris a spol. vo svojej práci [8] prezentujú dva algoritmy. Na nich ukazujú vzťah medzi počtom pátračov a časom potrebným pre dokončenie prehľadávania prostredia.

Tvorbe prehľadávacích stratégií pri oboch algoritmoch predchádza konverzia prostredia na graf použitím *triangulácie* (pre pripomenutie viď 3.4). Ak sú v grafe identifikované slučky, hľadajú sa uzly, ktorých vypustenie z grafu slučky odstráni. Vypustenie z grafu v tomto prípade znamená umiestnenie jedného pátrača do tohto uzlu pred začatím prehľadávania a jeho zotrvanie v ňom po celú dobu prehľadávania. Takýto pátrač sa nazýva *blokujúci*. Po umiestnení všetkých blokujúcich pátračov dostaneme z grafu strom, ktorý prehľadávame stratégiou vygenerovanou jedným z dvoch prezentovaných algoritmov.

4.4.1 Stručný prehľad

1. Dekompozícia pp na graf použitím triangulácie.
2. Konverzia grafu na strom umiestnením blokujúcich pátračov.
3. Aplikovanie prehľadávania do hĺbky alebo súčasného prehľadávania.

4.4.2 Prehľadávanie do hĺbky

Tento algoritmus je zameraný na použitie menšieho počtu pátračov, výmenou za dlhší čas prehľadávania.

Pred začatím prehľadávania je rekurzívne vypočítané označenie uzlov. Toto označenie reprezentuje počet pátračov, potrebný na prehľadanie podstromu s koreňom v aktuálnom uzle. Označenie pre všetky uzly vypočítame nasledujúcim algoritmom:

Po označení uzlov aplikujeme algoritmus 4, ktorý zaručí vyčistenie stromu.

Algoritmus 3 Algoritmus označenia uzlov

1: Každý list označ 1.
2: Potomkov každého uzlu zoraď podľa označenia od uzlu s najvyšším označením po uzol s najnižším označením.
3: **if** označenie prvého potomka nadradeného uzlu > označenie jeho druhého potomka **then**
4: Označ nadradený uzol rovnakým značením ako má prvý potomok.
5: **else if** označenie prvého potomka nadradeného uzlu = označenie jeho druhého potomka **then**
6: Označ nadradený uzol označením prvého potomka zvýšením o 1.
7: **end if**

Algoritmus 4 Algoritmus prehľadávania do hĺbky

1: Pošli potrebný počet pátračov do koreňového uzlu.
2: Pre všetkých jeho potomkov:
3: Ak sa jedná o list, pošli pátračov do nadradeného uzlu.
4: Ak sa nejedná o list, pošli potrebný počet pátračov na preskúmanie podstromu s koreňom v aktuálnom potomkovi a počkaj kým sa vrátia (= zavolaj algoritmus prehľadávania do hĺbky s použitím aktuálneho potomka ako koreň).
5: Ak sa jedná o posledný podstrom, pošli pátračov do nadradeného uzlu.

4.4.3 Súčasné prehľadávanie

Tento algoritmus je zameraný na čo najkratší čas prehľadávania, výmenou za väčší počet pátračov potrebných na jeho vykonanie.

Rovnako ako v predchádzajúcom algoritme sú všetky uzly označené počtom pátračov potrebných na ich prehľadanie. V tomto prípade to znamená, že každý uzol okrem listov (ktoré majú opäť označenie 1) je označený súčtom označení všetkých jeho potomkov.

Po označení uzlov aplikujeme algoritmus 5, ktorý zabezpečí vyčistenie stromu.

Algoritmus 5 Algoritmus súčasného prehľadávania

1: Pošli potrebný počet pátračov do koreňového uzlu.
2: Súčasne vyšli potrebný počet pátračov do všetkých potomkov.

4.5 Algoritmus s iteratívnym generovaním možných ciest

Kehagias a spol. prezentujú algoritmus [9] na prehľadávanie grafu reprezentujúceho pp, ktorý aplikuje viaceré vylepšenia na klasický algoritmus generovania všetkých možných stavov a následného hľadania cesty od počiatočného stavu k cieľovému. Tieto vylepšenia výrazne zvyšujú efektivitu algoritmu. Jedná sa napríklad o vypustenie stratégií, ktoré vedú k rovnakému stavu ako iná stratégia, no s menšou vyčistenou časťou prostredia. Algoritmus generuje stratégie prehľadávania s možnou rekontamináciou.

4.5.1 Algoritmus

Kvôli stručnosti a uľahčeniu pochopenia je algoritmus prezentovaný iba pre dvoch pátračov. Jeho rozšírenie pre väčší počet pátračov je zrejmé.

Pre pochopenie algoritmu je nutné najprv vysvetliť niekoľko pojmov.

Vstupmi algoritmu sú graf G a počiatočné pozície pátračov i, j. G je diskrétnou reprezentáciou pp získanou ľubovoľnou dekompozíciou. Skladá sa z množiny uzlov V a množiny hrán E. Počiatočné pozície pátračov sú čísla, identifikujúce konkrétne uzly z množiny V.

V algoritme používame pole trojíc (p, d, v). Na indexáciu tohto poľa slúži konfigurácia pátračov (unikátne rozmiestnenie pátračov do uzlov). Pre každú konfiguráciu teda udržiavame jednu trojicu:

p *Cesta* od počiatočnej pozície pátračov k aktuálnej konfiguráci. Napríklad pre dvoch pátračov má tvar $p^{n_1,n_2} = (i_1, i_2, ..., i_T; j_1, j_2, ..., j_T)$, kde i_1 až i_T je cesta pátrača n_1 a j_1 až j_T je cesta pátrača n_2.

d *Špinavá množina.* Jedná sa o bitový vektor, obsahujúci jeden bit pre každý uzol grafu G. Bity sú zoradené podľa označenia uzlov. Čistý uzol má označenie 0 a kontaminovaný (špinavý) 1.

v *Cena.* Je to súčet hodnôt v špinavej množine. Nadobúda hodnotu 0 práve vtedy, keď je vyčistené celé prostredie.

Zaujímavou charakteristikou algoritmu je spôsob reflektovania pohybu narušiteľov. Po každom ich ťahu je špinavá množina upravená tzv. *max-min násobením matíc.* Tento nástroj umožňuje jednoducho nastaviť rýchlosť narušiteľov na ľubovoľnú celo-číselnú kladnú hodnotu a zohľadniť ju pri úprave špinavej množiny. Detailnejší popis je uvedený na strane 6 v [9].

Kvôli uľahčeniu pochopenia je algoritmus 6 oproti pôvodnej podobe v [9] mierne zjednodušený.[1]

Keďže sa algoritmus neukončí kým nenájde úspešnú stratégiu, nie je jeho ukončenie garantované. Je preto pred jeho použitím vhodné zamyslieť sa nad obmedzením maximálneho počtu iterácií.

Kehagias a spol. prezentujú ešte jednu vylepšenú verziu algoritmu, ktorá funguje na podobnom princípe, no dosahuje lepšie výsledky z pohľadu časovej náročnosti. Okrem toho pridávajú ešte ďalšie vylepšenia efektívnosti algoritmu a heuristiku, ktorá zavádza stupeň náhodnosti do algoritmu. Táto heuristika zabráni "zasekávaniu sa" algoritmu v cestách, ktoré nevedú k vyčisteniu prostredia.

[1] Pre pôvodnú verziu viď [9] str. 10.

Algoritmus 6 Zjednodušená verzia algoritmu prezentovaného Kehagiasom a spol.

Vstup: Graf $G = (V, E)$; matica viditeľnosti; počiatočné pozície pátračov i, j
Inicializácia
Ku každej možnej konfigurácii pátračov priraď:
    (a) prázdnu cestu,
    (b) plnú špinavú množinu a
    (c) maximálnu cenu (N).
K počiatočnej konfigurácii priraď:
    (a) jednokrokovú cestu $\mathbf{p}^{(i,j)} = (i; j)$,
    (b) špinavú množinu $\mathbf{d}^{(i,j)}$ obsahujúcu všetky uzly mimo oblasti viditeľnosti a
    (c) zodpovedajúcu cenu $v^{(i,j)}$.
Najlepšiu cestu nastav na $\mathbf{p}^{(i,j)}$ a najlepšiu cenu na $v^{(i,j)}$.
Main
while najlepšia cena > 0 **do**
  for all všetky možné konfigurácie pátračov (n_1, n_2) **do**
    for all všetky možné nasledujúce konfigurácie (m_1, m_2) vychádzajúce z (n_1, n_2) **do**
      Vytvor novú cestu $\hat{\mathbf{p}} = (\mathbf{p}_1^{(n_1,n_2)}, m_1; \mathbf{p}_2^{(n_1,n_2)}, m_2)$, špinavú množinu $\hat{\mathbf{d}}$ a cenu $\hat{v}$.
      if nová cena $\hat{v} < v^{(m_1,m_2)}$ **then**
        Priraď k aktuálnej konfigurácii (m_1, m_2) novú cestu, špinavú množinu a cenu.
      end if
      if cena aktuálnej konfigurácie $<$ najlepšia cena **then**
        Nastav najlepšiu cestu na $\hat{\mathbf{p}}$ a najlepšiu cenu na $v^{(m_1,m_2)}$.
      end if
    end for
  end for
end while

4.6 Hollinger a spol.

Hollinger a spol. vo svojej práci [6] prezentujú niekoľko algoritmov. Spoločným znakom všetkých z nich je, že v každej iterácii produkujú jednu stratégiu prehľadávania. Kedykoľvek ich prerušíme, ich výstupom je najlepšia doteraz nájdená stratégia prehľadávania. Sú teda neterminujúce. Dobu ich vykonávania určí užívateľ napríklad časom ukončenia alebo nejakým požiadavkom vypočítanú na stratégiu. Požiadavkom môže byť napríklad počet dostupných pátračov.

V každej iterácii algoritmy generujú z grafu reprezentujúceho pp jeden vetviaci sa strom (*spanning tree* [2]). Z grafu vznikne strom umiestnením tzv. stacionárnych pátračov (nazývaných aj "stráž"). Stráže sú umiestnené do jedného z uzlov, medzi ktorými je hrana grafu nenachádzajúca sa vo vypočítanom strome. Pre ilustráciu, napríklad v prezentovaných porovnávacích testoch bolo generovaných stotisíc takýchto stromov.

Autori implementovali a porovnali tri metódy generovania vetviacich sa stromov. Jedná sa o spanning tree enumeration ([1]), uniform sampling ([22]) a algoritmus vyvinutý autormi práce - depth-first sampling (alg. 4 v [6]). Algoritmus DFS cielene uprednostňuje stromy, ktoré vyžadujú menej stacionárnych pátračov (vysvetlenie v nasledujúcom texte).

Pátrači používajú *implicitnú koordináciu* (pripomeňme si, že to znamená, že pred plá-

[2] http://en.wikipedia.org/wiki/Spanning_tree

novaním nasledujúceho kroku obdrží každý pátrač informácie o predchádzajúcich akciách všetkých ostatných pátračov a zahrnie ich do výpočtu svojej nasledujúcej akcie).

Po vytvorení vetviaceho sa stromu je na neho aplikovaný algoritmus označenia hrán ([6] alg. 2). Ten označí každú hranu číslom reprezentujúcim počet pátračov, ktorí ňou musia prejsť v smere od štartovného uzlu k listom pri exekvovaní prehľadávacej stratégie. Potom pridá ku každej hrane už spomínanú orientáciu od štartovného uzlu k listom a túto hranu zdvojí s hranou opačnej orientácie. Táto hrana opačnej orientácie dostane rovnaké označenie ale so záporným znamienkom. Bude reprezentovať vracajúcich sa pátračov.

Autori uverejňujú tri algoritmy na prechádzanie vytvorených vetviacich sa stromov - jednoduchý deterministický (zjednodušene popísaný tu 7[3]), vylepšený deterministický a náhodný.

Algoritmus 7 nespĺňa všetky spomínané vlastnosti, pretože je prezentovaný hlavne ako stavebný blok vylepšeného algoritmu (zjednodušene popísaného tu 8[4]).

Algoritmus 7 Jednoduchý algoritmus prechádzania vetviacich sa stromov prezentovaný Hollingerom a spol.

1: **Vstup:** Strom T; označenie hrán; počiatočný uzol (rovnaký pre všetkých pátračov)
2: Vypočítaj počet pátračov potrebných pre prehľadanie stromu.
3: **while** množina špinavých uzlov nie je prázdna **do**
4: Zvýš čas o 1.
5: **for all** množina pátračov **do**
6: **if** pátrač sa nemôže hýbať bez rekontaminácie **then**
7: Zostaň v aktuálnom uzle.
8: **else if** existuje kladne označená hrana priľahlá k aktuálnemu uzlu **then**
9: Presuň sa do uzlu, ku ktorému vedie hrana s najnižším kladným označením.
10: Zníž označenie tejto hrany o 1.
11: **else**
12: Presuň sa do uzlu so záporným označením.
13: Zvýš označenie tejto hrany o 1.
14: **end if**
15: Vyber z množiny špinavých uzlov všetky uzly, na ktorých sú momentálne pátrači.
16: **end for**
17: **end while**
18: **Výstup:** Stratégia prehľadávania a čas potrebný na jej uskutočnenie.

[3]Pre nezjednodušenú verziu viď algoritmus 3 v [6]
[4]Pre nezjednodušenú verziu viď algoritmus 5 v [6]

Algoritmus 8 Vylepšený algoritmus prechádzania vetviacich sa stromov prezentovaný Hollingerom a spol.

Vstup: Strom T; počiatočný uzol (rovnaký pre všetkých pátračov)
while neuplynul čas určený na exekúciu algoritmu **do**
  Vygeneruj vetviaci sa strom.
  Označ hrany vetviaceho sa stromu.
  Vypočítaj počet pátračov potrebných pre prehľadanie stromu.
  while graf nie je čistý **do**
    Pohni pátračov podľa algoritmu 7.
    if pátrač narazí na uzol U, ku ktorému prilieha ne-stromová (slučková) hrana **then**
      if nejaký stacionárny pátrač sa môže pohnúť bez rekontaminácie **then**
        Umiestni tohto stacionárneho pátrača do uzlu U.
      else
        Vygeneruj nového stacionárneho pátrača.
        Zvýš počet stacionárnych pátračov o 1.
      end if
    end if
  end while Zaznamenaj η =počet potrebných pátračov + počet stacionárnych pátračov.
end while
Výstup: Stratégia s najnižším η.

Kapitola 5

Pravdepodobnostné p-e algoritmy

5.1 P-e algoritmus s využitím A*

Moors a spol. vo svojej práci [15] prezentujú algoritmus, ktorý berie do úvahy obmedzenia senzorov používaných robotmi. Konkrétne sa jedná o možnosť neodhalenia narušiteľa nachádzajúceho sa v oblasti viditeľnosti pátrača. Modelujú ju pravdepodobnostnou funkciou, založenou na vzdialenosti narušiteľa od pátrača. Algoritmus predpokladá najhoršieho možného narušiteľa z pohľadu pátračov.

Pracovné prostredie je diskretizované *mrežovou diskretizáciou* (viď 3.3) s náhodným vyberaním bodov.

Pre každú bunku udržiavame *pravdepodobnosť* $P_{\bar{D}}$. $P_{\bar{D}}$ udáva pravdepodobnosť, s ktorou v nej nebol detekovaný narušiteľ. Pred začatím prehľadávania predpokladáme, že narušiteľ môže byť kdekoľvek. Preto inicializujeme $P_{\bar{D}}$ pre všetky bunky na 1.

Pre každú hranu definujeme *cenu*, definovanú ako čas, potrebný na jej prejdenie robotom.

Možný pohyb narušiteľov je modelovaný nastavením $P_{\bar{D}}$ každej konkrétnej bunky na hodnotu $P_{\bar{D}}$ bunky s najvyššou $P_{\bar{D}}$ spomedzi buniek z nej dosiahnuteľných pri danej rýchlosti narušiteľov.

Z dôvodu zníženia výpočtovej náročnosti je graf rozložený na niekoľko podgrafov umiestnením stacionárnych pátračov do niektorých uzlov grafu. Títo pátrači tvoria hranicu medzi podgrafmi. Výber vhodnej hranice je realizovaný vygenerovaním všetkých možných hraníc, ich ohodnotením heuristickou funkciou[1] a výberom najlepšie hodnotenej.

Prehľadávanie každého podgrafu je uskutočnené algoritmom A*[2]. Každý stav v prehľadávacom strome obsahuje:

- pozície všetkých pátračov (môžu byť v uzloch i hranách grafu prostredia) a
- $P_{\bar{D}}$ pre všetky uzly.

Hrany prehľadávacieho stromu tvoria akcie dostupné pátračom:

- výber nového cieľa (pátrač musí byť v uzle) a
- čakať na aktuálnej pozícii.

[1] Informácie o použitej heuristickej funkcii nájdete v kapitole 3, str. 3 [15].
[2] http://en.wikipedia.org/wiki/A*_search_algorithm

Počiatočný stav má začiatočné pozície pátračov (ktoré je možné vybrať ľubovoľne) a plne kontaminovaný graf. Koncový stav obsahuje pozície pátračov na konci prehľadávania a vyčistený graf.

Výber nasledujúceho stavu je riadený heuristickou funkciou dostupnou v [15] kapitola 3, str. 4. Efektívnosť algoritmu zvyšujú dve optimalizácie:

1. *vypustenie všetkých stratégií vedúcich k rekontaminácií* (Lapaugh v [13] dokázal, že tá nie je nutná pre nájdenie riešenia) a

2. *hashovanie stavov*, ktorým je dosiahnuté vypustenie stavov rovnakých až na permutáciu pátračov. Napríklad stav v ktorom ide pátrač 1 na pozíciu A a pátrač 2 na pozíciu B je rovnaký, ako stav, v ktorom ide pátrač 1 na pozíciu B a pátrač 2 na pozíciu A. Stačí teda ponechanie jedného z nich.

Nakoniec je nutné stratégie pre jednotlivé podgrafy zlúčiť do jednej prehľadávacej stratégie. To je dosiahnuté pridaním jednoduchých presunov pátračov z pozícií, v ktorých sú na konci prehľadávania jedného podgrafu do počiatočných pozícií prehľadávania nasledujúceho podgrafu. Jednoduché sú preto, lebo nehrozí rekontaminácia, keďže kontaminované časti grafu sú oddelené hranicou medzi podgrafmi.

5.1.1 Stručný prehľad

1. Aplikuj mrežovú diskretizáciu na pp.

2. Rozdeľ graf do podgrafov.

3. Pre každý podgraf definuj počiatočný a koncový stav.

4. Na každý podgraf aplikuj A* s použitím definovaného počiatočného a koncového stavu.

5. Zlúč prehľadávace stratégie pre podgrafy do jednej.

5.2 Online pravdepodobnostný algoritmus s nastaviteľnými narušiteľmi

Strom a spol. vo svojej práci [19] prezentujú algoritmus, ktorý pracuje na princípe *maximalizácie odmeny*. Pod odmenou v tomto prípade rozumieme pravdepodobnosť dolapenia narušiteľov. Algoritmus pracuje s grafom reprezentujúcim pp. Prehľadávanie tohto grafu prebieha generovaním *prehľadávacieho stromu* a jeho prechádzaním. Prechádzanie prehľadávacieho stromu je riadené spomenutou snahou o maximalizáciu získaných odmien.

Hľadanie vhodnej stratégie pre viacero pátračov naraz je výpočtovo veľmi náročný problém. Túto náročnosť ešte zvyšuje predpoklad nepriateľských narušiteľov. Autori algoritmu prinášajú tri spôsoby, ako sa s ňou vysporiadať:

1. vlastný *spôsob modelovania narušiteľov* (bližšie popísaný v 5.2.2),

2. použitie *sekvenčného plánovania* pre pátračov a

3. hľadanie *suboptimálnej stratégie*.

Sekvenčné plánovanie Pri tomto type plánovania si každý pátrač samostatne určí svoj nasledujúci plán s použitím "starých" plánov poskytnutých ostatnými pátračmi. Potom pátrači vykonajú jeden krok, vymenia si svoje plány a opäť plánujú nasledujúci plán.

Takýto prístup negarantuje optimálnu stratégiu, no zníži vetvenie prehľadávacieho stromu z a^p na a, kde a je počet akcií dostupných pátračom a p je počet pátračov. Viac informácií o sekvenčnom plánovaní je možné nájsť napríklad v [5], prípadne ohľadom podobného iteratívneho prístupu v [16].

Hľadanie suboptimálnej stratégie Užívateľ zadá hodnotu parametra ϵ. Tento parameter hovorí, o koľko musí byť maximálna odmena získateľná nejakou trajektóriou väčšia, ako odmena doposiaľ najlepšej trajektórie, aby sme pokračovali v jej generovaní. Týmto môžme výrazne znížiť počet generovaných trajektórií. Pre viac informácií viď [19] kapitola 3, časť A.

5.2.1 Algoritmus

Ako už bolo spomenuté, algoritmus generuje prehľadávací strom pre pátračov. Tento strom sa vetví s každou vykonanou akciou (napr. presun do susedného uzlu grafu pp). Každý uzol stromu má priradenú pravdepodobnosť dolapenia narušiteľa (tiež známu ako *odmena*) R. Pri tvorbe trajektórie cez prehľadávací strom (t.j. plánu pre jedného pátrača) sa odmena kumuluje na základe princípu popísaného v kapitole 3, časť A. Každý pátrač si pri výbere trajektórie zvolí tú, ktorá má najvyššiu R. Hĺbka prehľadávania stromu (t.j. dĺžka trajektórií spomedzi ktorých si pátrač vyberá tú s najvyššou R v krokoch) je nepriamo určená užívateľom, keď zadá hodnotu už spomenutého parametra ϵ.

Pátrači si udržujú pravdepodobnosť výskytu každého narušiteľa v každom uzle grafu pp a kumulatívnu pravdepodobnosť jeho dolapenia $P(dolapenie)$. Tieto pravdepodobnosti sa menia na základe akcií pátračov a modelu narušiteľov. Keď pátrač prejde cez uzol, pravdepodobnosť výskytu každého narušiteľa v tomto je zakomponovaná do zodpovedajúcej $P(dolapenie)$ a pre tento uzol je nastavená na 0. Vplyv modelu narušiteľov na $P(dolapenie)$ je ten, že do nej zakomponujeme P z rovnice 5.1.

$$P = P_{cr}^{S} \tag{5.1}$$

P vyjadruje pravdepodobnosť, že všetky trajektórie vygenerované narušiteľom vedú k jeho dolapeniu. P_{cr} je pravdepodobnosť dolapenia narušiteľa pri použití *modelu s náhodným rozptylom*[3]. Pre viac informácií viď [19] kapitola 3, časť B. S je úroveň zručnosti narušiteľov (viď 5.2.2).

Pomocou $P(dolapenie)$ a rovnice 3 v [19] pátrači vypočítajú odmeny spojené s jednotlivými akciami a teda dokážu rozhodovať o najvhodnejšej trajektórii.

5.2.2 Model narušiteľa

Zručnosť narušiteľa je vyjadrená parametrom S. Tento parameter určí užívateľ. V prípade že ho užívateľ nepozná, autori prezentujú v kapitole 5 spôsob, akým ho môžu pátrači odhadnúť.

Narušitelia si v tomto modele generujú rovnaký prehľadávací strom ako pátrači, no namiesto pravdepodobnosti dolapenia narušiteľa v ňom majú pravdepodobnosť úniku. Hodnota S určuje, koľko listov takéhoto stromu narušiteľ vygeneruje. Generovanie týchto listov prebieha náhodne. Po ich vygenerovaní si narušiteľ zvolí trajektóriu podľa listu s najvyššou pravdepodobnosťou úniku. Čím vyššiu hodnotu teda parameter S má, tým má narušiteľ

[3]Pri tomto modele sa pravdepodobnosť, že sa narušiteľ nachádza v konkrétnom uzle po každom jeho pohybe rovnomerne rozloží na všetky uzly v jeho dosahu.

väčší výber listov a teda tým je "zručnejší". Ak je rovný 1, jedná sa o náhodne sa pohybujúceho narušiteľa.

Kapitola 6

Implementácia appletu

Mojou úlohou bolo vytvoriť java applety prezentujúce niektoré algoritmy popisované v tejto práci. Cieľom bolo tiež vytvoriť framework pre jednoduché pridávanie ďalších p-e algoritmov a dekompozícií. V tejto kapitole stručne popíšem hierarchiu a funkciu najpodstatnejších tried, schopnosti appletov a tiež ich ovládanie.

Pre umožnenie práce s appletom a jeho rozširovania aj ľuďom nehovoriacim slovensky či česky som sa rozhodol text v applete i zdrojový kód a dokumentáciu písať v anglickom jazyku.

Momentálne je implementovaný iba applet pre Kehagiasov algoritmus popísaný v 4.5.

Applet sa skladá z ovládacích prvkov, plochy na vykresľovanie priebehu algoritmu a stavového riadku.

6.1 Ovládanie appletu

Na začiatku je applet vo fáze prípravy pp (pridávanie prekážok, robotov a pod). Po odštartovaní simulácie je vykresľovaný jej priebeh. Najprv dekompozícia pp na graf a následne prezentácia vypočítanej prehľadávacej stratégie.

6.1.1 Príprava pracovného prostredia

Prípravu pp je možné preskočiť vybraním jedného z pripravených pp zo zoznamu. V prípade tvorby vlastného pp či úpravy niektorého z pripravených je nutné pridať prekážky, pátračov a narušiteľov. Tiež je nutné vybrať jednu z dekompozícií. Tieto úkony sú intuitívne, takže nebudú ďalej popisované. Vykonávanie ktorejkoľvek akcie v príprave pp je možné zrušiť kliknutím pravým tlačítkom myši do plochy pre vykresľovanie. Je nutné spomenúť, že v súčasnej verzii appletu sú prekážky obmedzené na *jednoduché polygóny* a *nesmú sa navzájom prekrývať*. Momentálne je implementovaná jedna dekompozícia, a to dekompozícia pomocou obmedzenej Delaunayho triangulácie (3.5).

Je tiež možné nastaviť rýchlosť narušiteľov. Tá môže byť v rozmedzí jedna až počet uzlov grafu. Rýchlosť rovná počtu uzlov grafu prakticky vyjadruje nekonečnú rýchlosť v rámci daného grafu.

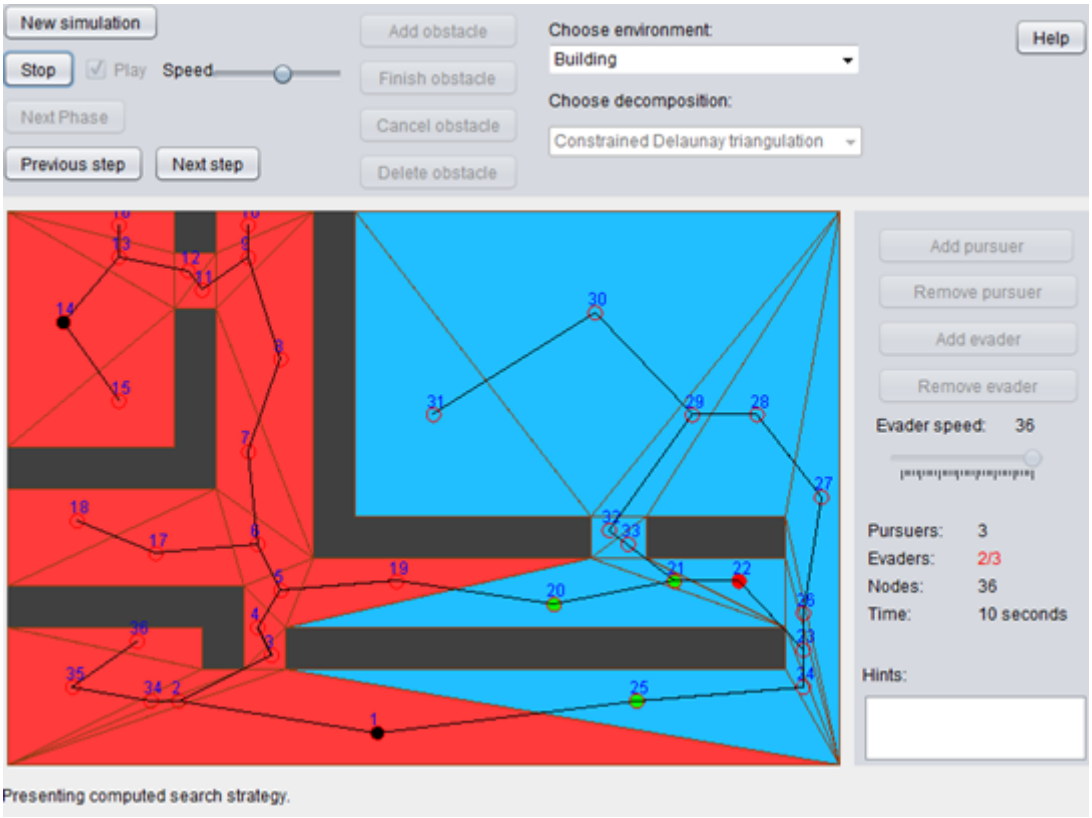
Počet uzlov grafu vzniknutého dekompozíciu pp, počet pátračov i počet narušiteľov je uvedený v pravom spodnom rohu appletu. Z dôvodu vysokej výpočtovej náročnosti implementovaného algoritmu je odporúčaný maximálny počet pátračov päť a maximálny počet uzlov dvadsaťpäť. Pre väčšinu navrhnutých prostredí by mali stačiť traja až štyria pátrači. Počet narušiteľov výpočtový čas neovplyvňuje.

Pri vložení alebo odstránení prekážky sú odstránení všetci doteraz vložení roboti, keďže bol zmenený graf a uzly v ktorých boli umiestnení už nemusia existovať..

Nakoniec je možné vybrať si medzi automatickým prehrávaním simulácie, alebo manuálnou navigáciou pomocou rozhrania. V prípade výberu automatického prehrávania je možné nastaviť jeho rýchlosť.

6.1.2 Simulácia

Od začiatku spustenia simulácie beží v pravej dolnej časti časovač. Je podľa neho napríklad možné odsledovať dĺžku výpočtu prehľadávacej stratégie.



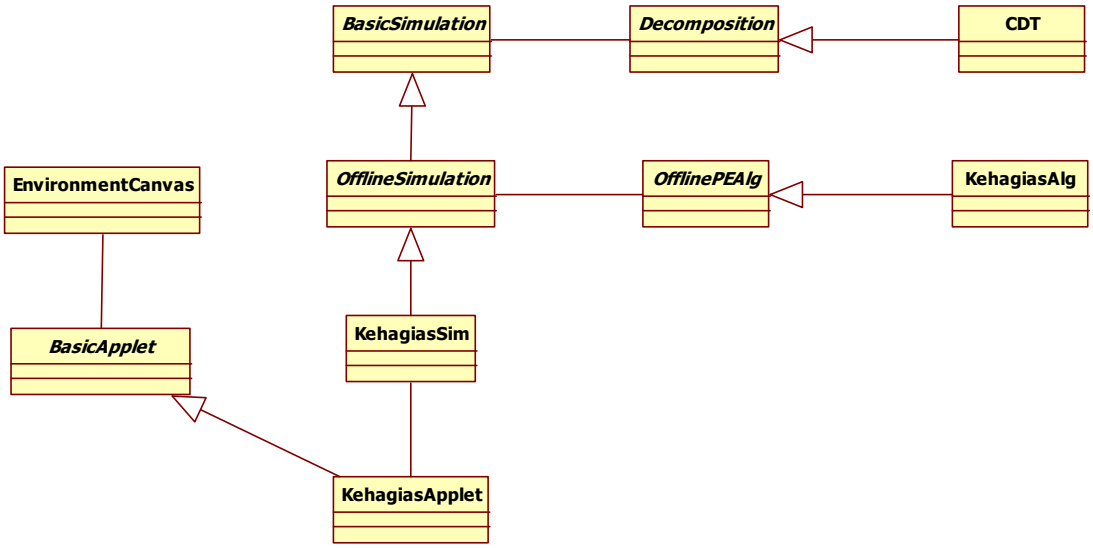
Obrázek 6.1: Applet vo fáze simulácie.

Simuláciu je možné kedykoľvek pozastaviť a znovu spustiť tlačítkom *Stop* respektíve *Start*. Zrušiť ju je možné tlačítkom *New simulation*.

Pri prezentovaní vypočítanej prehľadávacej stratégie počet narušiteľov indikuje aj počet doteraz nechytených narušiteľov.

Pohyb narušiteľov je náhodný, ale pohybujú sa iba v kontaminovanej časti prostredia.

6.2 Hierarchia tried



Obrázek 6.2: UML diagram najdôležitejších tried appletu.

Hierarchia najpodstatnejších tried je zobrazená na obrázku 6.2.

- **Hlavná trieda appletu**, *KehagiasApplet*, rozširuje abstraktnú triedu *BasicApplet*, ktorá poskytuje väčšinu potrebnej funkcionality. Pridáva ovládacie prvky, ktoré nie sú prítomné v ostatných appletoch.
- **Základná trieda pre implementáciu appletu**, *BasicApplet*, implementuje základné užívateľské rozhranie a ovládanie appletu.
- **Trieda vykresľujúca pp a priebeh algoritmu**, *EnvironmentCanvas*, poskytuje prostriedky pre prácu s pp. Tiež vykresľuje priebeh algoritmu.
- **Triedy implementujúce simuláciu** - *Simulation, OfflineSimulation, KehagiasSim.* Tieto triedy zabezpečujú riadenie simulácie. Trieda pre simuláciu online p-e algoritmov ešte nie je implementovaná.
- **Triedy implementujúce dekompozíciu pp** - *Decomposition, CDT.* Abstraktná trieda *Decomposition* popisuje základné požiadavky na konkrétnu implementáciu dekompozície. Trieda *CDT* implementuje dekompozíciu pomocou obmedzenej Delaunayho triangulácie (3.5).
- **Triedy implementujúce p-e algoritmus** - *OfflinePEAlg, KehagiasAlg.* Základná trieda pre online p-e algoritmy ešte nebola implementovaná. Trieda *KehagiasAlg* je bližšie popísaná v časti 6.3.
- Ďalej applet obsahuje veľké množstvo pomocných tried navrhnutých s cieľom uľahčenia rozširovania appletu.

6.3 Implementácia Kehagiasovho algoritmu

Pri implementácii tohto algoritmu bolo vytvorených viacero tried, ktoré bude možné využiť pri implementácii ďalších p-e algoritmov (napr. trieda pre prácu s konfiguráciou pátračov *Keh_pursConfig* či trieda pre prácu s prehľadávacou stratégiou *Keh_pathList*).

Hlavnou časťou triedy *KehagiasAlg* je metóda *execute()*. Tá vykoná algoritmus hľadania prehľadávacej stratégie a vráti ju v podobe triedy *TimeSteps*. S tou ďalej pracuje trieda ovládajúca simuláciu *KehagiasSim*. Pred zavolaním metódy *execute()* stačí vytvoriť inštanciu triedy *KehagiasAlg* a zavolať jej metódu *init()*.

Hlavnou dátovou štruktúrou algoritmu je trieda *Keh_triplesMap*, ktorá asociuje konfigurácie pátračov s trojicou cesta, špinavá množina a cena.

Implementovaná je optimalizácia pomocou uchovávania iba jednej z permutácií každej možnej konfigurácie pátračov.

6.4 Kroky pri tvorbe appletu pomocou implementovaného frameworku

1. Implementácia p-e algoritmu odvodením z triedy *OfflinePEAlg*.[1]
2. Implementácia simulačnej triedy odvodenej z triedy *OfflineSimulation*.[2] Tá zahŕňa hlavne vytvorenie metódy *run()* riadiacej beh simulácie.
3. Odvodenie appletu od triedy *BasicApplet*. Pri tom je nutné vytvoriť zoznam dekompozícií, ktoré je možné použiť spolu s daným p-e algoritmom. Voliteľným krokom je pripravenie prostredí prezentujúcich činnosť algoritmu.

6.5 Použité knižnice

Pri implementácii boli využité nasledovné knižnice:

1. *jgrapht-jdk1.6* - pre prácu s grafmi.
2. *colt* - pre prácu s binárnymi maticami.
3. *poly2tri-core-0.1.1* - knižnica implementujúca obmedzenú Delaunayho trianguláciu.

[1] V prípade implementácie appletu pre online p-e algoritmus je nutné najprv implementovať základnú triedu *OnlinePEAlg*.

[2] V prípade implementácie appletu pre online p-e algoritmus je nutné najprv implementovať základnú triedu *OnlineSimulation* odvodenú od triedy *Simulation*.

Kapitola 7

Záver

V rámci bakalárskej práce som zmapoval práce dostupné v oblasti p-e algoritmov, vytvoril framework pre tvorbu appletov prezentujúcich p-e algoritmy, implementoval v ňom jeden z p-e algoritmov a jednu dekompozíciu pp. Vytvorené boli tiež stránky prezentujúce informácie o jednotlivých algoritmoch a v prípade Kehagiasovho algoritmu i vytvorený applet.

Pri skúmaní oblasti p-e algoritmov som zistil, že sa jedná o veľmi rozsiahlu oblasť, v ktorej je zverejnené obrovské množstvo prác. Ako zrejme všetky oblasti súvisiace s robotikou, i táto sa v súčasnosti rozvíja viac ako kedykoľvek v minulosti. Jedným z možných smerov rozširovania tejto práce do budúcnosti je teda snaha držať krok s týmto rozvojom, aby mohla záujemcom v koncentrovanej a názornej forme prezentovať aktuálne vyvíjané p-e algoritmy. Dobrým zdrojom najnovších výsledkov v tejto oblasti je každoročná konferencia ICRA.

Kvôli rozsiahlosti oblasti nebolo možné implementovať všetko čo som chcel. I preto som si dal za cieľ vytvoriť už spomenutý framework, s ktorého využitím by na moju prácu mohol v budúcnosti nadviazať niekto ďalší. Väčšina v súčasnosti dostupných algoritmov je offline, takže pri tvorbe frameworku som sa zameral práve na tvorbu appletov pre offline p-e algoritmy. Pridanie podpori pre online algoritmy je ďalšou vhodnou témou na rozšírenie práce.

Literatura

[1] Char, J.: Generation of Trees, Two-Trees, and Storage of Master Forests. *Circuit Theory, IEEE Transactions on*, ročník 15, č. 3, sep 1968: s. 228 – 238, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1082817.

[2] Choset, H.; Lynch, K. M.; Hutchinson, S.; aj.: *Principles of Robot Motion: Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents)*. The MIT Press, Červen 2005, URL: http://books.google.cz/books?id=S3biKR21i-QC&printsec=frontcover&hl=cs#v=onepage&q&f=falsen.

[3] Gerkey, B. P.; Thrun, S.; Gordon, G.: Visibility-based Pursuit-evasion with Limited Field of View. *Int. J. Rob. Res.*, ročník 25, č. 4, Duben 2006: s. 299–315, URL: http://www.cs.cmu.edu/~ggordon/gerkey-thrun-gordon.ijrr06.pdf.

[4] Guibas, L. J.; Latombe, J.-C.; Lavalle, S. M.; aj.: A Visibility-Based Pursuit-Evasion problem. *International Journal of Computational Geometry and Applications*, ročník 9, 1999: s. 471–494, URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.6928&rep=rep1&type=pdf.

[5] Hollinger, G.; Djugash, J.; Singh, S.: Coordinated search in cluttered environments using range from multiple robots. In *in Proc. 6th Int'l Conf. on Field and Service Robotics (FSR '07)*, 2007, URL: http://www.ri.cmu.edu/pub_files/pub4/hollinger_geoffrey_2007_2/hollinger_geoffrey_2007_2.pdf.

[6] Hollinger, G.; Kehagias, A.; Singh, S.: GSST: anytime guaranteed search. 2010, URL: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=B8FF7B2D42FD77254CD1C556A0D64B0B?doi=10.1.1.174.2576&rep=rep1&type=pdf.

[7] Isler, V.; Kannan, S.; Khanna, S.: Randomized pursuit-evasion in a polygonal environment. *Robotics, IEEE Transactions on*, ročník 21, č. 5, oct. 2005: s. 875 – 884, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1512346.

[8] Katsilieris, F.; Lindhé, M.; Dimarogonas, D. V.; aj.: Demonstration of Multi-Robot Search and Secure. In *IEEE International Conference on Robotics and Automation*, 2010, URL: http://www.ee.kth.se/~lindhe/katsilieris2010dmr.pdf.

[9] Kehagias, A.; Hollinger, G.; Singh, S.: A Graph Search Algorithm for Indoor Pursuit / Evasion. Technická Zpráva CMU-RI-TR-08-38, Robotics Institute, Pittsburgh, PA, August 2008, URL: http://www.ri.cmu.edu/pub_files/pub4/kehagias_athanasios_2008_2/kehagias_athanasios_2008_2.pdf.

[10] Kolling, A.: *Multi-robot Pursuit-Evasion*. Dizertační práce, University of California, Merced, December 2009, URL: https://robotics.ucmerced.edu/Robotics/papers/AndreasKollingPhDThesis.pdf.

[11] Kolling, A.; Carpin, S.: Surveillance strategies for target detection with sweep lines. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, oct. 2009, s. 5821 –5827, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5354225.

[12] Kvasnica, M.: *Vizualizace hledání cesty pro robota,*. Bakalářská práce, FIT VUT v Brně, 2008, URL: http://www.stud.fit.vutbr.cz/~xkvasn03/bakalarka/xkvasn03.pdf.

[13] LaPaugh, A. S.: Recontamination does not help to search a graph. *J. ACM*, ročník 40, April 1993: s. 224–245, URL: http://doi.acm.org/10.1145/151261.151263.

[14] LaValle, S.; Hinrichsen, J.: Visibility-based pursuit-evasion: the case of curved environments. *Robotics and Automation, IEEE Transactions on*, ročník 17, č. 2, apr 2001: s. 196 –202, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=928565&userType=inst.

[15] Moors, M.; Rohling, T.; Schulz, D.: A probabilistic approach to coordinated multi-robot indoor surveillance. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, aug. 2005, s. 3447 – 3452, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1545038.

[16] Nair, R.; Tambe, M.; Yokoo, M.; aj.: Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings. In *In IJCAI*, 2003, s. 705–711, URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.6512&rep=rep1&type=pdf.

[17] Rodriguez, S.; Denny, J.; Burgos, J.; aj.: Toward realistic pursuit-evasion using a roadmap-based approach. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, may 2011, s. 1738 –1745, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5980467.

[18] Shewchuk, J. R.: Delaunay Refinement Algorithms for Triangular Mesh Generation. *Computational Geometry: Theory and Applications*, ročník 22, 2001: s. 1–3, URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.19.2501&rep=rep1&type=pdf.

[19] Strom, J.; Morton, R.; Reilly, K.; aj.: Online Probabilistic Pursuit of Adversarial Evaders. In *IEEE International Conference on Robotics and Automation (ICRA2010) Workshop*, 2010, URL: http://www.frc.ri.cmu.edu/~gholling/SearchWorkshop/papers/Strom_icra10_spe.pdf.

[20] Vieira, M. A. M.; Govindan, R.; Sukhatme, G. S.: Scalable and Practical Pursuit-Evasion. In *Second International Conference on Robot Communication and Coordination*, March 2009, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4957471.

[21] Volkov, M.; Cornejo, A.; Lynch, N. A.; aj.: Environment Characterization for Non-recontaminating Frontier-Based Robotic Exploration. In *PRIMA'11*, 2011, s. 19–35, URL: http://www.springerlink.com/content/w8882695p4420303/fulltext.pdf.

[22] Wilson, D. B.: Generating Random Spanning Trees More Quickly than the Cover Time. In *Proceedings of the twenty-eighth annual acm symposium on the theory of computing*, ACM, 1996, s. 296–303, URL: http://citeseer.ist.psu.edu/viewdoc/download;jsessionid=1D6C27F8C77908300BAEAB19D40E7870?doi=10.1.1.47.8598&rep=rep1&type=pdf.

Příloha A

Obsah CD

CD priložené k tejto práci obsahuje:

- zdrojový kód appletu a frameworku pre tvorbu appletov (java),
- zdrojový kód výukových stránok (HTML, PHP 5), stránky sú tiež prístupné na adrese http://www.stud.fit.vutbr.cz/~xtomek07/bp,
- zdrojový kód textovej časti bakalárskej práce vo formáte LaTeX a
- pdf prác s tématikou riešenia problému pursuit-evasion nájdených pri vypracovávaní práce.